Juan Aja Fernandez,
Stephan Wenger,
Christophe Morisset,
Marcus Magnor

Algebraic 3D Reconstruction of Planetary Nebulae

Technical Report / Computer Graphics Lab, TU Braunschweig ; 2008-6-7

Veröffentlicht: 01.07.2008

http://www.digibib.tu-bs.de/?docid=00022781

Juan Aja Fernández
juan.aja@gmail.com

Stephan Wenger
swenger@gmx.net

Dr. Christophe Morisset
chris.morisset@gmail.com
Instituto de Astronomia, UNAM, Mexico

Prof. Dr. Ing. Marcus Magnor
magnor@cg.tu-bs.de
Computer Graphics Lab, TU Braunschweig

Algebraic 3D Reconstruction of Planetary Nebulae

Technical Report 2008-8-7

21. August 2008

Computer Graphics Lab, TU Braunschweig

Abstract

Distant astrophysical objects like planetary nebulae can normally only be observed from a single point of view. Assuming a cylindrically symmetric geometry, one can nevertheless create 3D models of those objects using tomographical methods. Small deviations from axial symmetry can be corrected by means of heuristic processes, though the arising 3D models are, in general, no longer unambiguously defined. Making use of *Volume Rendering* techniques, the created models are then visualized.

Weit entfernte astrophysikalische Objekte wie planetarische Nebel sind der Beobachtung in der Regel nur aus einem einzelnen Blickwinkel zugänglich. Unter Annahme einer zylindersymmetrischen Geometrie lassen sich dennoch mit tomographischen Verfahren 3D-Modelle solcher Objekte erzeugen. Kleinere Abweichungen von der Zylindersymmetrie lassen sich mittels heuristischer Methoden im Modell ergänzen, die entstehenden 3D-Modelle sind jedoch im Allgemeinen nicht mehr eindeutig. Mittels *Volume Rendering* werden die entstandenen Modelle visualisiert.

Contents

1 Introduction **1**
1.1 Planetary Nebulae 1
1.2 Tomographic Reconstruction 1
1.3 The Problem of Astrophysical Observations 3

2 Related Work **5**
2.1 Tomographic Reconstruction of Flames 5
2.2 CIVR for Planetary Nebulae 5
2.3 Doppler Shift Methods 6

3 Theoretical Considerations **7**
3.1 Specifying the Linear System 7
3.2 Solving the Linear System 8
3.3 Finding the Right Inclination 9
3.4 Correcting for Asymmetries 9
3.5 Visualizing the Results 11

4 Implementation **13**
4.1 Specifying the Linear System 13
4.2 Solving the Linear System 14
4.3 Correcting for Asymmetries 16
4.4 Visualizing the Results 16

5 Results **19**
5.1 Mz3 (Ant Nebula) 19
5.2 NGC6543 (Cat's Eye Nebula) 24
5.3 NGC7009 (Saturn Nebula) 26

6 Discussion and Conclusion **29**
6.1 Discussion 29
6.2 Conclusion 30
6.3 Outlook 30

7 Acknowledgments **31**

Chapter 1

Introduction

1.1 Planetary Nebulae

When stars not larger than a few sun masses die, they often eject part of their matter until only a small glowing nucleus is left in the center of a gaseous shell. These objects are called *planetary nebulae* (PN).[1]

Even though the typical diameter of a planetary nebula is about one light-year, they are usually not visible to the naked eye because of their low brightness. Their shape is often spherical (fig. 1.1) or bipolar (i.e. cylindrically symmetric, fig. 1.2), but irregular shapes exist as well.

The shell of a planetary nebula consists mainly of hydrogen and helium, but other elements like oxygen and neon are also present. Due to the radiation of the central star, these atoms get ionized and begin to emit light of characteristical wavelengths when the electrons recombine. This makes it easy to discern planetary nebulae from clouds of dust that would show absorption lines instead of emission lines.

If the radiation of the central star suffices to ionize the whole shell, its visible shape is determined by the presence of matter – the nebula is *matter bounded*. In the contrary case, it is called *radiation bounded*. The unionized part is then not visible and cannot easily be determined.

For a more comprehensive introduction to planetary nebulae, see [OF06].

1.2 Tomographic Reconstruction

A common approach for getting three-dimensional volume models from two-dimensional images is *tomographic reconstruction*[KS88]. This method is used, for example, in *computed tomography* (CT) to get volume densities out of multiple x-ray images of an object. While in the case of CT images

[1] When planetary nebulae were first observed through telescopes, they received their name because of their resemblance to gas planets – they have nothing to do with planets apart from that.

Figure 1.1: The Spirograph Nebula, or IC418, an example for a roughly spherically symmetric planetary nebula. From `http://hubblesite.org/gallery/album/entire_collection/pr2000028a/`

Figure 1.2: The Ant Nebula, or Mz3, an example for a roughly cylindrically symmetric planetary nebula. From `http://hubblesite.org/gallery/album/entire_collection/pr2001005a/`

the density of the object causes *absorption*, the contrary is the case for planetary nebulae, which emit light proportionally to the density of ionized gas (neglecting the absorption of the nebula in the wavelengths that are used for reconstruction). The intensity I_i of a certain pixel i in a discrete two-dimensional image of a planetary nebula is just the integral over all the emission densities along the incident light ray of this pixel. Using a discrete volume model, this can be written as a sum over all volume elements v_j, where each summand is the length of the ray that lies inside the volume element, multiplied by its emission density ρ_j:

$$I_i = \sum_j |\mathrm{Ray}(i) \cap v_j| \cdot \rho_j \tag{1.1}$$

This system of linear equations, usually written as $A\mathbf{x} = \mathbf{b}$ with $x_j = \rho_j$, $b_i = I_i$ and $A_{ij} = |\mathrm{Ray}(i) \cap v_j|$, can now be solved for the ρ_j. Under certain preconditions, this gives a unique solution for the volume emission densities.

For the solution to be uniquely defined, the rank of the matrix A must be at least equal to the number of volume elements. This is usually achieved by using images from multiple viewpoints. In practice, the system will almost always be overdetermined, as one would rather use more pixels than necessary to get more stable results. An approximate solution can then be computed using iterative algorithms that minimize the residual error $||A\mathbf{x} - \mathbf{b}||_2$.

1.3 The Problem of Astrophysical Observations

For most astrophysical objects, getting images from multiple viewpoints is impossible due to the large distance to those objects. This means that if a regular grid of volume elements (or *voxels*) is used, the linear system is not uniquely defined and would in fact not yield any information about the three-dimensional structure of the object.

This problem can be solved by making additional assumptions about the geometry of the object, e.g. an axial symmetry that is common in planetary nebulae. Such symmetries can easily be reflected in the choice of voxels by combining all regions of space that are assumed to have the same emission density into one voxel (fig. 1.3). This can reduce the three-dimensional complexity of the voxels (x, y, z) to a two-dimensional one in the case of cylindrical voxels (r, z) so that the solution of the linear system is unique.

While symmetries are common in planetary nebulae, their symmetry is never perfect. To get more realistic results, the residual pixel intensities $\mathbf{b} - A\mathbf{x}_{\mathrm{approx}}$ can be distributed among the voxels that contribute to the intensity of the corresponding pixel. Because no depth information is available for these unmatched emissivity densities, the distribution among the voxels is not uniquely defined and can in fact only be chosen using heuristic methods.

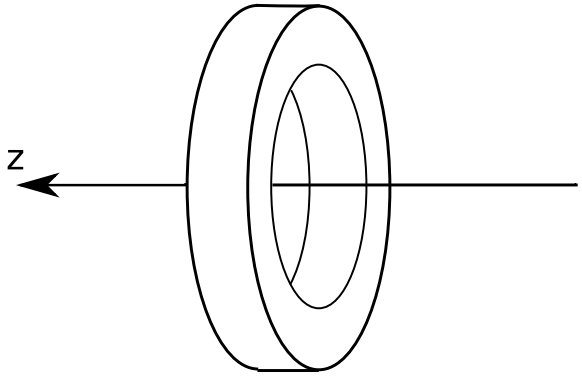

Figure 1.3: A voxel representing the axial symmetry of the geometry. Since all locations that lie in the same voxel share the same intensity, using voxels of this form guarantees axial symmetry of the result.

In general, influences of the light path between the object and the observer also have to be considered. Interstellar dust, for example, might non-homogeneously absorb light. Since the ratio of the intensities of some spectral lines tends to be the same for all planetary nebulae but dust absorption is usually wavelength-dependent, the amount of absorption can be estimated when calibrated images for multiple wavelength are available. In our cases, absorption outside the nebula seems to be mostly negligible, so no correction is needed.

Chapter 2

Related Work

2.1 Tomographic Reconstruction of Flames

In classical tomography, usually the absorption of a volume is measured from different directions. This causes nonlinearities in the formula of the pixel intensities because absorption is usually not only proportional to the volume density but also to the ray's intensity. In our case, however, a linear formula applies because the accumulated intensity along the ray does not influence the emission at another place. Ihrke and Magnor[IM04] implemented an algebraic reconstruction approach for exactly this case. However, as they reconstruct the non-symmetric geometry of flames, their algorithm does not take symmetry into account, but relies on images taken from multiple viewpoints that are not available in the case of astronomical objects.

2.2 Constrained Inverse Volume Rendering for Planetary Nebulae

In [MKHD04] and [MKHD05], Magnor et al. developed a reconstruction method for planetary nebulae that is based on *Constrained Inverse Volume Rendering* and the assumption of axial symmetry. While they propose corrections for small deviations from axial symmetry, these corrections are not realized in the provided examples. Furthermore, our approach is outperforming the CIVR algorithm by far; while using CIVR "the reconstruction of a 128x32-pixel density map takes approximately one day on a 2.4 GHz PC in conjunction with an nVidia GeForce FX 3000 graphics card", we can reconstruct a nebula with comparable resolution in a matter of seconds, asymmetry correction included, if the correct inclination angle is known. Otherwise, the reconstruction process has to be repeated for a range of possible inclinations, but the whole reconstruction will still not take more than a few minutes.

2.3 Doppler Shift Methods for Retrieving Depth Information

Sabbadin et al.[SCB+00] take a totally different approach for the reconstruction of the 3D geometry of planetary nebulae. They assume that the velocity of a certain region of gas around the nebula is strongly correlated to its distance from the central star. Calculating the Doppler shift of some well-known emission lines allows to get the velocity component towards the observer. Combining these, depth information can be reconstructed. However, the relation between velocity and distance from the central star is generally not straightforward to determine, and exact Doppler shift measurement requires elaborate experimental setups, while our reconstruction approach relies on easily available photographic images only.

Chapter 3

Theoretical Considerations

3.1 Specifying the Linear System

In order to specify the system of linear equations (eq. 1.1), we need to calculate the matrix elements A_{ij} and the right-hand vector $\mathbf{b}$. While the b_i are already given by the intensities of the pixels, setting up the A_{ij} requires further calculations.

We recall that $A_{ij} = |\mathrm{Ray}(i) \cap v_j|$ is the length of the ray through pixel i that lies inside the volume element j. This means we have to calculate the intersection points of lines with voxels that have the form of a hollow cylinder (cf. fig. 1.3). These calculations are somehow simplified by the fact that since the object in question is far away (and relatively small, so that the viewing angle is small), we can assume that all the incident light rays are orthogonal to the image plane, which allows us to use rays of the form

$$\mathbf{r}(t) = \begin{pmatrix} p_x \\ p_y \\ -t \end{pmatrix}, \tag{3.1}$$

where p_x and p_y are the pixel x and y coordinates, respectively. This means we are using a three-dimensionally enhanced version of the image coordinate system for our calculations.

Due to their axial symmetry, the voxels are naturally specified in cylinder coordinates of a coordinate system whose z-axis is parallel to the symmetry axis of the nebula and whose origin lies in its center (i.e. the position of the central star).[1] Thus, every voxel is defined by its minimum and maximum radius $(r_{\min}, r_{\max})$ and z-axis intercepts $(z_{\min}, z_{\max})$. The whole cylindrical geometry is specified by the position of the central star in pixel coordinates, the direction of the symmetry axis (angle with respect to the x-axis and

[1] Note that, because of the parallel light rays, the distance between the observer and the nebula is of no importance to the algorithm and the cylinder origin can in fact lie in the image plane.

inclination with respect to the image plane), the length and diameter of the nebula (in pixels) and the desired resolution of the model, that is the number of slices along the symmetry axis and the number of rings perpendicular to it.

The part of a ray that lies inside a voxel can be regarded as the difference of the length of the ray that lies inside its outer cylinder slice of radius $r_{\max}$ and the length that lies inside its inner cylinder slice of radius $r_{\min}$, so that only an algorithm for intersection of z-axis-aligned rays with arbitrarily oriented cylinders is needed. Such algorithms are widely in use and can be found in libraries like [geo].

3.2 Solving the Linear System

The linear system $A\mathbf{x} = \mathbf{b}$ must now be solved for the voxel intensities $\mathbf{x}$. Since the system is usually overdetermined, in general only an approximate solution minimizing the residual norm $||A\mathbf{x} - \mathbf{b}||_2$ is possible. This solution can be determined by iterative algorithms such as the *Conjugate Gradient Least Squares* (CGLS) method.

The fundamental idea of the Conjugate Gradient algorithm is that solving $A\mathbf{x} = \mathbf{b}$ with A symmetric and positive definite is equivalent to minimizing $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}A\mathbf{x} - \mathbf{b}^{\mathrm{T}}\mathbf{x}$. Starting from $\mathbf{x} = 0$, each iteration step k modifies the intermediate solution vector $\mathbf{x}_k$ by descending in the direction of the gradient of $f(\mathbf{x})$, so that $\mathbf{x}_{k+1} = \mathbf{x}_k + \epsilon_k \nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k}$. For fast convergence, it is important that ϵ_k is carefully chosen, and that the directions of descent are *conjugate* to each other, that means that for all directions $\mathbf{d_i} = \nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_i}$, $\mathbf{d_j} = \nabla \mathbf{f}(\mathbf{x})|_{\mathbf{x}=\mathbf{x_j}}$: $\mathbf{d_i}^{\mathrm{T}} A \mathbf{d_j} = 0$.

In our case, however, the matrix A does not fulfill the above conditions. But since for any matrix A the matrix product $A^{\mathrm{T}}A$ is symmetric and positive definite, we can multiply our system $A\mathbf{x} = \mathbf{b}$ by A^{T} and solve $A^{\mathrm{T}}A\mathbf{x} = A^{\mathrm{T}}\mathbf{b}$ instead. The CGLS implementation does this multiplication implicitly, preserving sparsity of the matrix A, which allows for more efficient (memory saving) algorithms.

Since the norm $||\mathbf{x}||_2$ of the intermediate solutions increases monotonically during the iteration (see p. 75 of [Han96]), it is necessary to start the iteration with $\mathbf{x} = 0$ in order to not exclude any possible solution. The residual norm, on the other hand, is guaranteed to decrease monotonically, so convergence is guaranteed if numerical errors can be neglected. In practice, the iteration can be stopped as soon as the convergence speed falls below a chosen minimal value.

In the original algorithm, the value range for the vector $\mathbf{x}$ is not restricted in any way. Particularly, the entries of the solution vector can be negative. Since negative emission intensities are impossible (they cannot even be regarded as a physically valid model for absorption), the intermediate solutions

have to be projected onto the subspace of positive solutions after each step so that the positivity of the solution is guaranteed (cf. [IM04]).

After this step, we have obtained a *radial map* of the model, that is a 2D grid of densities whose axes are the r and z cylinder coordinates of the corresponding voxel (cf. fig. 5.2). Rotating this map around the z axis gives the full axisymmetric 3D model.

3.3 Finding the Right Inclination

While the rotation of the projection of the nebula's axis inside the image plane is usually easy to determine for the human observer, the inclination of the axis with respect to the image plane is difficult to tell from the image. If no additional information is available, the inclination must somehow be determined automatically. For this, a best-fit approach could be used: the reconstruction process would be run with different inclination values (from a user specified range), and the result that leaves the least residual would be kept. This is mostly equivalent to finding the inclination that allows for the largest axial symmetry, and should thus be close to the real inclination.

However, the comparison of residual norms must be done with care. In fact, the convergence speed of the CGLS algorithm varies widely with different inclination angles, and to be able to use a meaningful stopping criterion one would need to estimate the final residual at the same time, which turns out to be a highly nontrivial task. We therefore have to rely on previous computations for the inclination angles.

A simpler way to determine the inclination can be used for nebulae that have circular, ring-like features perpendicular to their symmetry axis: The circle, when projected, becomes an ellipse, and from the ratio of both semi-axes one can determine the inclination as

$$\cos(\phi) = \frac{\text{semiminor axis}}{\text{semimajor axis}}. \tag{3.2}$$

Because of the orthogonal projection that emerges of our assumption that the nebula is very far away compared to its size and because of the fact that intensities along a ray are only summed up and do not influence each other, there is one inherent ambiguity in this approach: every positive inclination cannot be distinguished from the negative inclination of the same amount. But as this ambiguity is also present in the following step, the final model can be easily mirrored at the image plane to get the equivalent result for negative inclinations.

3.4 Correcting for Asymmetries

While from a macroscopic point of view many planetary nebulae show axial symmetry, on a smaller scale there is always some deviation from perfect

symmetry. This can be seen in the residual image that is left when the projection of the reconstructed model onto the image plane is subtracted from the real image. We will now attempt to distribute this residual intensity among the voxels of our model so that the projection equals the original image (cf. fig. 5.7, fig. 5.8 and 5.9).

Since there is no depth information available for the asymmetric part of the intensity, this distribution will always be ambiguous, so we have to choose a "distribution function" that gives subjectively good results when viewed from different angles.

To break up the axial symmetry, the model of cylindrical voxels is first converted into a model of cubic voxels. For simplicity, the model will be aligned to the image plane, so that its x and y resolution equal the image x and y resolution, while the z resolution of the model must be chosen so that the whole cylindrical model lies inside the newly generated one. While this generates a lot of "empty" cubic voxels, it simplifies the following calculations dramatically.

In the next step, the intensity for each cubic voxel is calculated by casting a ray through the pixel on which it is projected and intersecting all the cylindrical voxels again. The resulting partial ray that lies inside a certain cylindrical voxel is also intersected with the cubical voxel to get the amount of intersection among both voxels along the viewing ray.[2] This amount is then multiplied by the intensity of the cylindrical voxel, and all those intensities are summed up to get the intensity for the whole cubic voxel.

Now that we have converted the cylindrically symmetric volume model into one that allows for asymmetry, we may think about a way to distribute our residual intensities to the voxels. Since by the choice of our cubic voxel model every voxel only contributes to a single pixel, we do not need to take interdependencies between voxels into account. So the only decision that is left is which voxels that share common x and y coordinates will get how much of the residual intensity of the pixel (x, y).

A convenient distribution function seems to be the following: Each pixel gets an amount of residual intensity that is proportional to the amount of intensity it already contributes to the pixel intensity. So if the reconstructed pixel intensity is $I_p = \sum_{v \in V} I_v$ where V is the set of all voxels that contribute to the current pixel, the residual intensity is I_r and the original pixel intensity is $I_o = I_p + I_r$, the new voxel intensities are

$$I'_v = I_v + \frac{I_v}{I_p} I_r = I_v \left(1 + \frac{I_r}{I_p}\right) = I_v \left(\frac{I_o}{I_p}\right). \tag{3.3}$$

As this is just a multiplication of all voxels that are projected onto the same pixel with a common value $\frac{I_o}{I_p}$, this is efficient to calculate knowing the

[2] A not so accurate but much faster way to solve this would be to calculate for each cubic voxel the cylindric voxel in which its center pixel lies and to set the intensity of the cubic voxel to the intensity of this cylindric voxel.

projected intensity I_p and the residual I_r, and can also easily be parallelized if necessary. The function is also optimal in the sense that it guarantees non-negative voxel intensities whenever possible, which in our case is always the case because the pixel intensities are always non-negative. It also preserves visual coherence between neighboring voxels (which usually have similar intensities).

While this approach yields good results as long as the asymmetries are small compared to the intensities, large residuals may cause all the voxels along one line of sight to become very bright or dark, which is physically very unplausible. An approach that handles this case better is to distribute the intensity only to the brightest voxel in front of and behind the cylinder axis plane. This is based on the assumption that asymmetries only occur on the outer part of the shell (due to collisions with interstellar gas clouds, for example) and that the shell is thin so that the radius of maximum intensity is a good approximation for its position. When this is not the case, it would be possible to use a threshold to find the maximum radius that is still considered to lie inside the shell. Even a more sophisticated contour detection that guarantees smooth results would be possible.

3.5 Visualizing the Results

The output of the reconstruction algorithm is a three-dimensional grid of cubic voxels, each of which has a certain emission density. To visualize this grid from an arbitrary viewpoint, *volume ray casting* can be used. This means that from each screen pixel a ray is cast through the volume and intensities along the ray are summed up. Since the voxel grid is dense and regular, no sophisticated and computationally expensive intersection tests are needed, but one can simply trace the ray from its grid entry point to the point where it leaves the grid just looking for adjacent voxels and summing up their densities.

In order to get an impression of the chemical composition of a planetary nebula, reconstructed voxel models for different wavelengths can be shown simultaneously, assigning a color to each of them. Since the interesting spectral lines are often too close to each other (like the hydrogen and nitrogen lines at 656nm and 658nm, respectively) or even invisible to the human eye, false-color display is used. In the simplest case, when three different source images are to be displayed, these are naturally assigned to the red, green and blue color channels.

The geometric nature of the problem and the independence between calculations for different screen pixels make this an ideal candidate for implementation on graphics hardware. As we will see in the next chapter, it is indeed possible to do volume ray casting in real time using shader programming.

Chapter 4

Implementation

4.1 Specifying the Linear System

The most important part of the calculations needed to specify the system of linear equations is the calculation of intersections between rays and voxels. The core algorithm for intersection between a line and a cylinder is adapted from [geo]. Since the rays are assumed to be parallel to each other (cf. eq. 3.1) and the cylindrical voxels share their axis of symmetry, a number of optimizations have been possible.

Firstly, the coordinate system of the cylinder (i.e. a coordinate system whose z-axis is parallel to the cylinder axis) can be precomputed independently for all rays and voxels. The origin of this coordinate system is assumed to be the center of the cylinder slice that is intersected, so it has to be recalculated for voxels that have different z coordinates. Most other computations depend on the cylinder origin, so in order to keep the number of recalculations low, the iteration over different voxel z coordinates is carried out in the outermost loop.

Then, the ray is transformed to the cylinder coordinate system, which allows for easier intersection tests. Since this transformation has to be done once per pixel and per cylinder origin, the results can be reused for different voxel radii. So the innermost loop has to be the iteration over different voxel r values. Additionally, the matrix element A_{ij} is calculated as the difference of intersections with the outer and inner cylinders of the voxel j, but since one voxel's outer cylinder is the inner cylinder of the voxel with the next bigger radius, those values only need to be computed once as long as the innermost loop iterates over those different radii in order.

While parallelization of this algorithm would be possible because the matrix elements are essentially independent, care must be taken not to lose the benefits of precomputing common values. Naively parallelizing the inner loop, for example, would double the amount of computations needed because intersection values for adjacent voxels cannot be reused. But parallelizing

this loop (possibly using CUDA[cud]) would probably still speed up the overall process, so this function stays a candidate for optimization.

The only important calculation to be done for the vector b is to correct for any nonlinearities in the image creation process. This is important because the specification of the linear system requires b to contain values proportional to the integral of intensities along the incident rays, while for human viewing a logarithmic scale is better suited. Since the astrophysical images we use are usually not postprocessed, we can just use the unscaled pixel values for b.

4.2 Solving the Linear System

Implementations of iterative least-squares solvers for linear systems are widely available (Matlab's `x=A\b` operation, for example), but normally do not allow for additional restrictions to the solution vector $\mathbf{x}$. Since we need to guarantee $x_j \geq 0$ for all vector components, we modify an existing Matlab implementation[han] and translate it to C.

For the matrix manipulations that are used in the algorithm (mainly vector sums, vector-matrix-multiplications and vector norms) several libraries are available on the net, most of which descended from the somewhat classical Fortran BLAS library[bla]. We used a CPU-based BLAS implementation, called UBLAS[ubl], which is part of the well-known boost library, and a GPU-based variant, CUBLAS from nvidia[cud].

The same algorithm, implemented once using UBLAS and once using CUBLAS, shows great differences in runtime (fig. 4.1). The GPU-based, parallelized variant is about 100 times faster than the CPU-based approach. However, the matrix size in the CUBLAS version is limited by the graphics card memory and the maximum texture size. While taking advantage of the sparsity of A would reduce the memory consumption (the matrix A is usually very sparse), sparse matrix storage is unfortunately not implemented in CUBLAS, so the size of the linear system seems to be unchangeably limited.

Knowing that the size of matrix A is $n_{\text{slices}} \cdot n_{\text{rings}} \cdot w \cdot h \cdot 4$ where w and h are the input image width and height, respectively, and 4 is the number of bytes per matrix element, and further assuming that the nebula covers the whole input image (which can be achieved by rotating and cropping) and that we have about 4 pixels for each voxel, we can calculate the required memory M:

$$M \approx (wh)^2 \,\text{bytes} = (4 n_{\text{slices}} n_{\text{rings}})^2 \,\text{bytes}, \tag{4.1}$$

so that our limitation of $M < 768\text{MB}$ leads to $n_{\text{slices}} n_{\text{rings}} < 7000$, that is, for example, models with 120 slices and 60 rings. Where this limit is not acceptable, the much slower UBLAS approach has to be used.

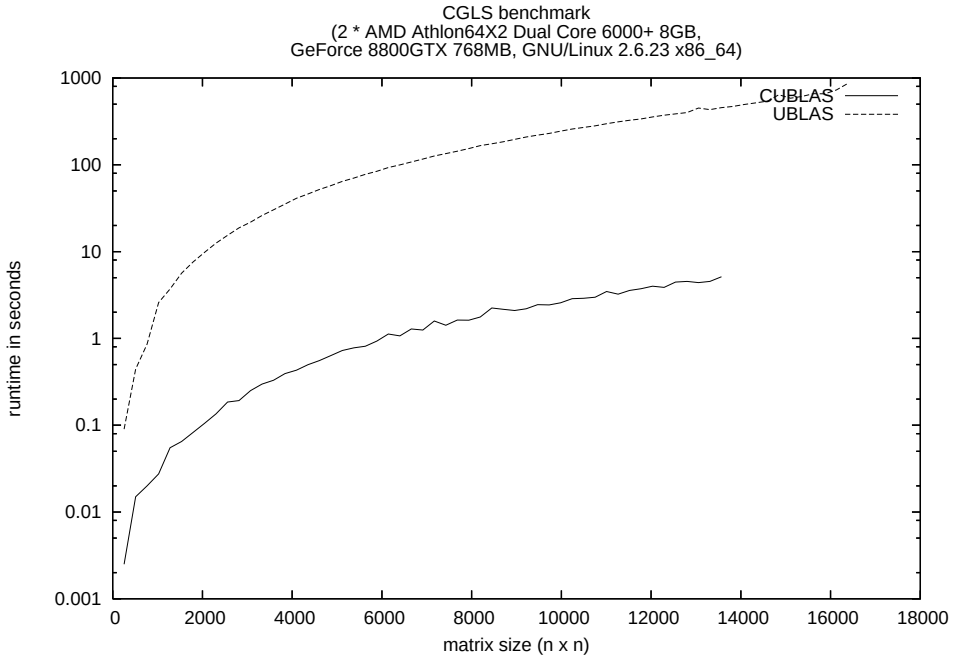

Figure 4.1: Comparison of computing time for UBLAS and CUBLAS implementations, in logarithmic scale. For large matrices, the GPU-based computation is about a hundred times faster than the CPU-based approach, but matrix size is limited by the available GPU memory (here: 768MB $\approx 14000 \cdot 14000 \cdot 4$B for 4-byte floating point numbers).

4.3 Correcting for Asymmetries

The asymmetry correction mainly consists of two parts: The calculation of intersections between cylindrical and cubic voxels, which can be simplified by intersecting both of them with a ray instead, and the distribution of the residual.

Since the distribution of the residual only iterates over all cubic voxels, its runtime is negligible compared to the intersection calculation which iterates over all cubic and cylindrical voxels. The operations used inside the corresponding loop are also cheap (cf. eq. 3.3), so that its implementation is straightforward.

The calculation of intersections or rather overlaps between cylindrical and cubic voxels can be simplified if one recalls that earlier we assumed that pixels get their whole intensity from the ray that hits their center. This not only allows to only regard the overlap of cylindrical and cubic voxel along that ray, but also to reuse the cylinder intersection values computed during the matrix creation (cf. section 4.1). Since all rays are parallel to the z axis, the intersection of a ray (that is now limited by its intersection with a cylindrical voxel) with a cubic voxel is reduced to comparing the z coordinates of the limiting voxel planes with the intersection t coordinates and selecting the strictest limiting values.

Though the calculations that are needed for the intersection calculation are quite simple, the sheer number of voxels causes quite a long computing time. We therefore also implement the (faster, but less accurate) alternative approach: for each cubic voxel, we calculate its center and the cylindrical voxel in which the center lies. We then just set the cubic voxel's intensity to the intensity of the corresponding cylindrical voxel.

4.4 Visualizing the Results

As we have seen in the previous chapter, the volume ray casting process is well suited for implementation on graphics hardware. The voxel model, for example, can easily be represented as a three-dimensional texture. Integrating the intensities along a given ray can then be approximated by just summing up the texture values at a number of fixed-distance sampling points along the ray.

A more sophisticated model could be used to get exact results by intersecting the ray with the cubic voxels and summing up the intensities multiplied with the length of the ray that lies inside the corresponding voxel. This could be efficiently implemented by traversing the grid using the 3DDDA algorithm described in [FTI88].

The sum of intensities along a ray can be efficiently calculated in parallel for all rays using a simple fragment shader. The entry and exit point to the

volume are most easily determined by rendering a cuboid of appropriate dimensions – whose vertices reflect their x, y and z coordinates in their RGB color values or 3D texture coordinates – twice, once using back face culling and once using front face culling, and using those RGB or texture coordinate values inside the fragment shader.

Chapter 5

Results

In order to generate realistic 3D models from the available data, we need to do some preprocessing. First, the image is thresholded so that all the pixels considered to be "background" are set to zero. Then, stars other than the central star have to be removed because they are almost certainly not within the volume we want to visualize and thus would cause artifacts in the model. This is done by creating a mask (by hand) of image parts that are influenced by non-nebula light sources. The masked regions are then filled by diffusion to get a smooth image: Regions of the image that are not masked keep their intensity, while masked pixels are iteratively filled so that their intensity $B(x, y)$ fulfills $\Delta B = 0$ (a Poisson equation with fixed border conditions). The vanishing of the Laplacian guarantees maximal smoothness in the masked area.

The inclination of the nebula with respect to the image plane can either be estimated by looking for circular shapes and measuring their distortion, or the reconstruction process can be carried out for multiple inclination angles and the result with the least residual norm is kept (with the restrictions discussed in section 3.3).

5.1 Mz3 (Ant Nebula)

The Ant Nebula, discovered in 1922 by Donald Menzel, has a number of different gaseous outflows from its bright center. The most visible outflow consists of two approximately spherical lobes, but more subtle "rays" can also be observed outside these lobes. Interestingly, all these features share a common axis of symmetry, so in principle the simultaneous reconstruction of all important features would be possible. However, due to the large difference in intensity and the linear scale chosen in the visualization, these structures cannot be observed together in one output image. Anyway, the interesting fine-grained asymmetries of the spherical lobes are visible in the output.

Due to the presence of many bright stars in the original images, heavy preprocessing needed to be done. This may have caused loss of fine-grained structures in some areas.

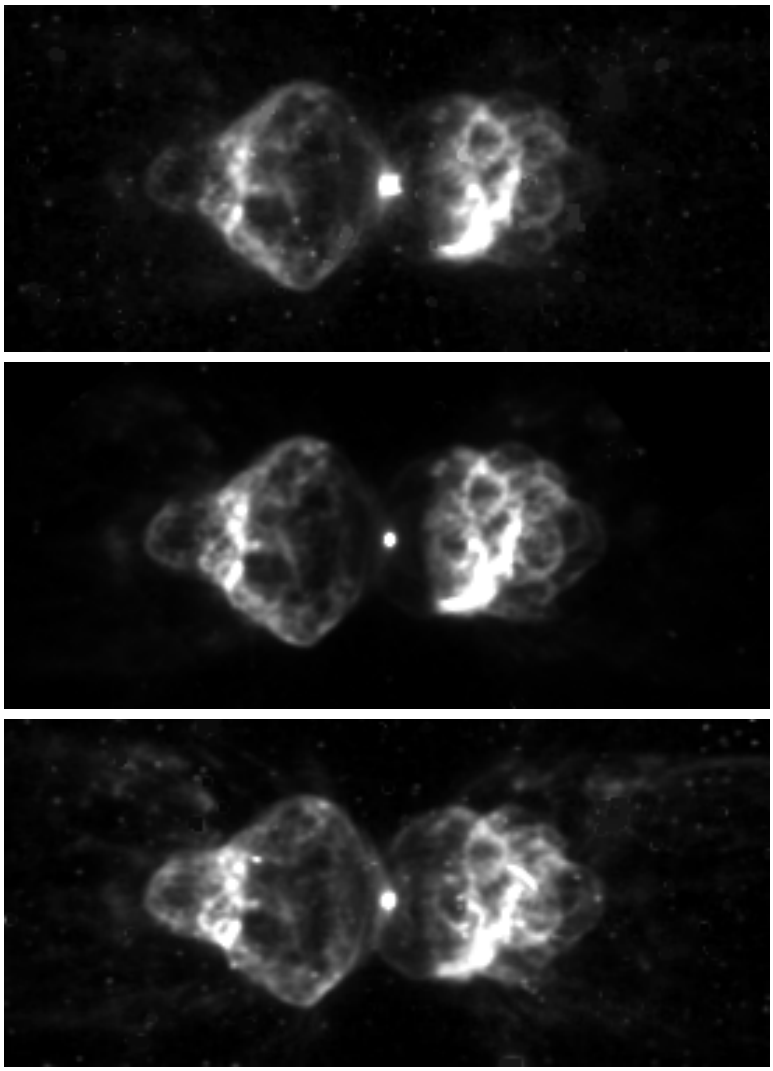

Figure 5.1: The Ant Nebula, images taken using filters for 487nm, 658nm and 673nm, respectively.

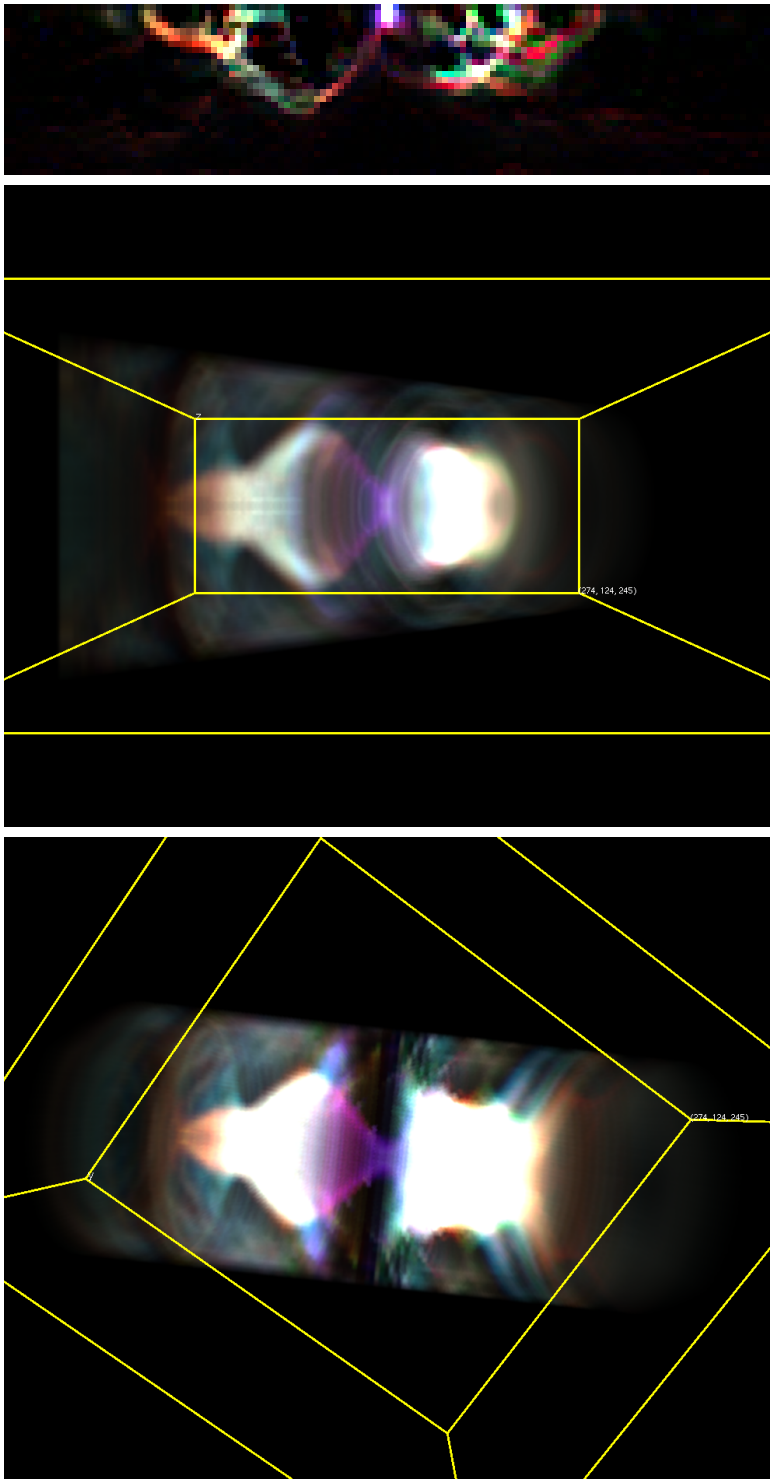

Figure 5.2: Reconstruction of the Ant Nebula, using 35° inclination: radial map (horizontal axis: z coordinate, vertical axis: r coordinate, with 0 at top) and reconstructed view from earth and from outer space. The red, green and blue color channels are assigned to 673nm, 658nm and 487nm, respectively.

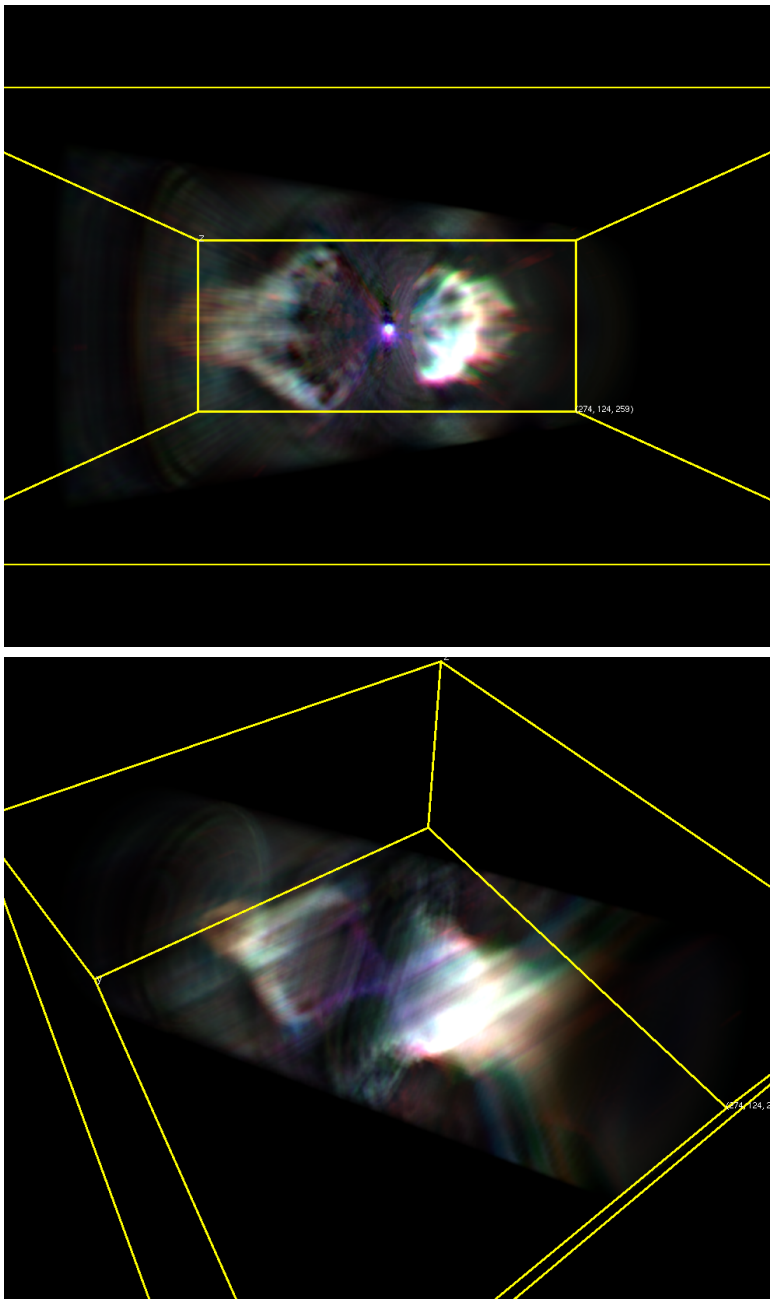

Figure 5.3: Asymmetry correction, using the proportional intensity rule, for the Ant Nebula: view from earth and from outer space. The red, green and blue color channels are assigned to 673nm, 658nm and 487nm, respectively. In the second image, artifacts of the asymmetry correction can be seen as lines parallel to the viewing direction of the original image.

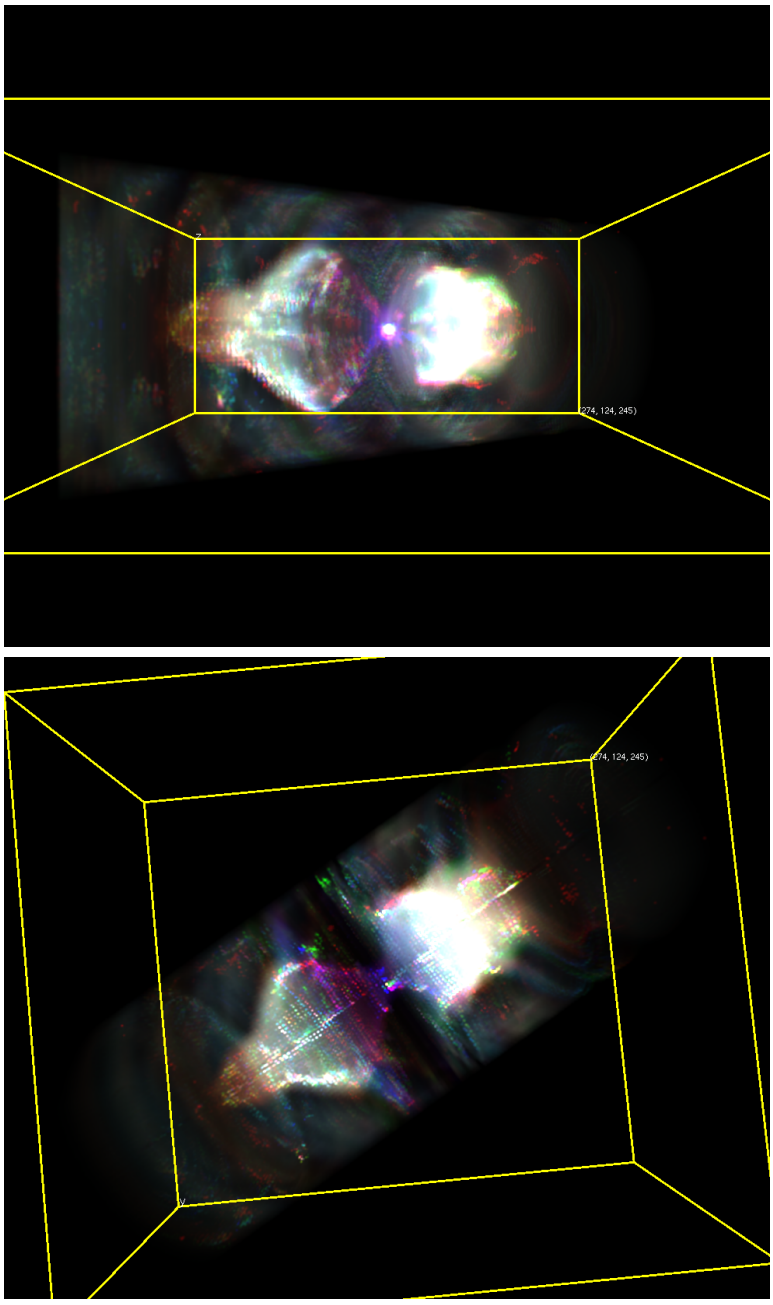

Figure 5.4: Asymmetry correction, using the maximum intensity rule, for the Ant Nebula: view from earth and from outer space. The red, green and blue color channels are assigned to 673nm, 658nm and 487nm, respectively. Note how the visual continuity of these images is affected by the choice of the distribution function, while the line artifacts disappear.

5.2 NGC6543 (Cat's Eye Nebula)

The Cat's Eye Nebula, discovered by William Herschel in 1786, has a very complex and not quite axisymmetric structure. Its reconstruction is nevertheless quite accurate due to the asymmetry correction. Due to its relatively large brightness compared to the surrounding stars, no preprocessing was needed to get clean results.

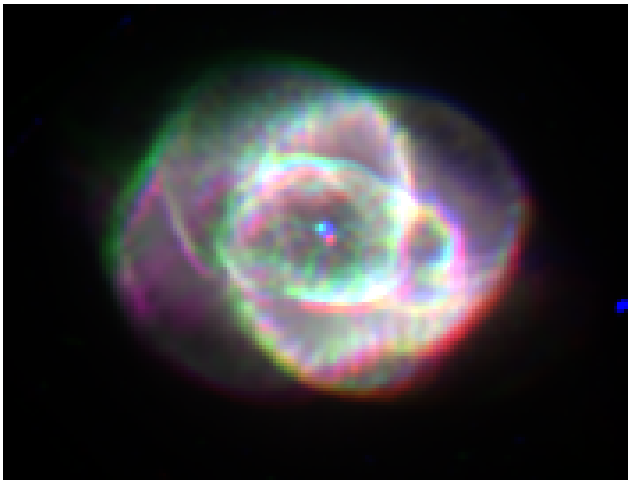

Figure 5.5: The Cat's Eye Nebula, red, green and blue channels assigned to images taken using filters for 487nm, 502nm and 656nm, respectively.

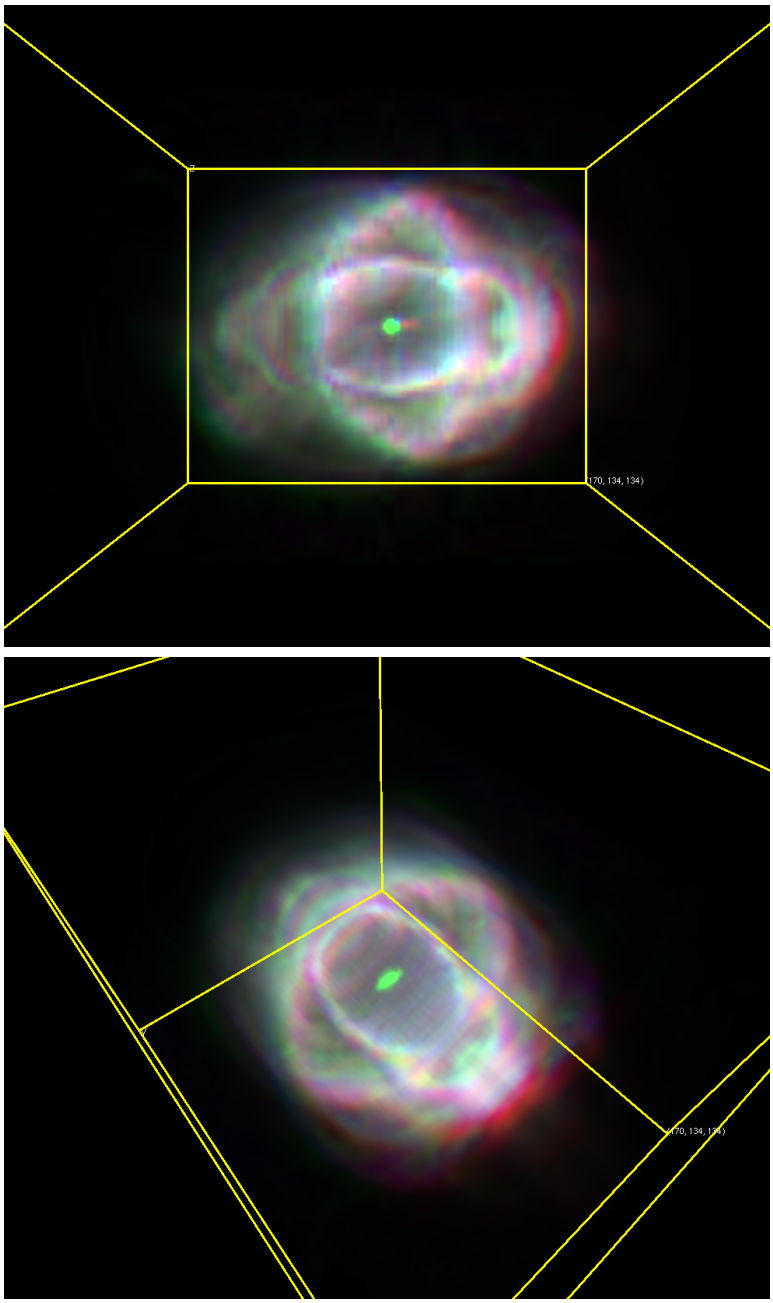

Figure 5.6: Asymmetry correction, using the proportional density rule, for the Cat's Eye Nebula: view from earth and from outer space. The red, green and blue color channels are assigned to 656nm, 502nm and 487nm, respectively.

5.3 NGC7009 (Saturn Nebula)

The Saturn Nebula, discovered by William Herschel in 1782, shows a bright, slightly S-shaped structure in the center, surrounded by a darker, barrel-shaped one. The S-shaped structure has noticeable reddish glowing tips. The original images were moderately disturbed by stars, so slight preprocessing had to be done.

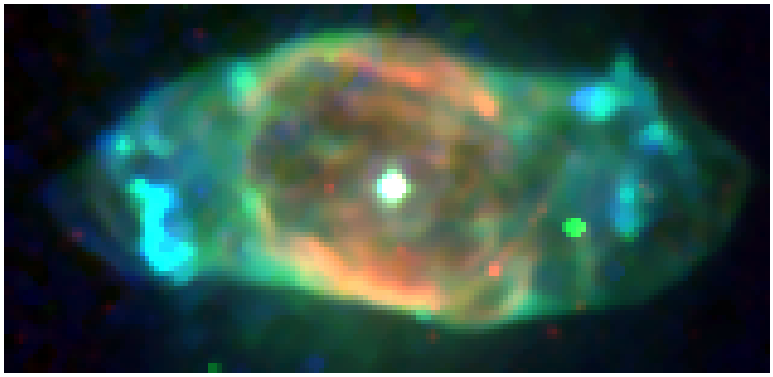

Figure 5.7: The Saturn Nebula, red, green and blue channels assigned to images taken using filters for 502nm, 555nm and 658nm, respectively.

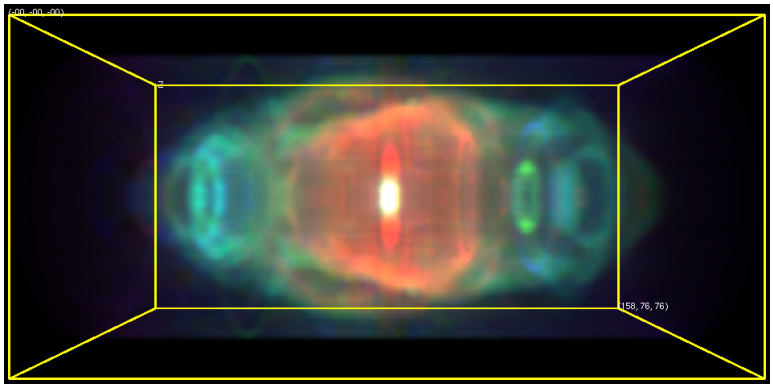

Figure 5.8: Reconstruction of the Saturn Nebula without asymmetry correction: view from earth and from outer space. The red, green and blue color channels are assigned to 658nm, 555nm and 502nm, respectively. The S-shape of the original image is lost because it violates the request for axisymmetry.

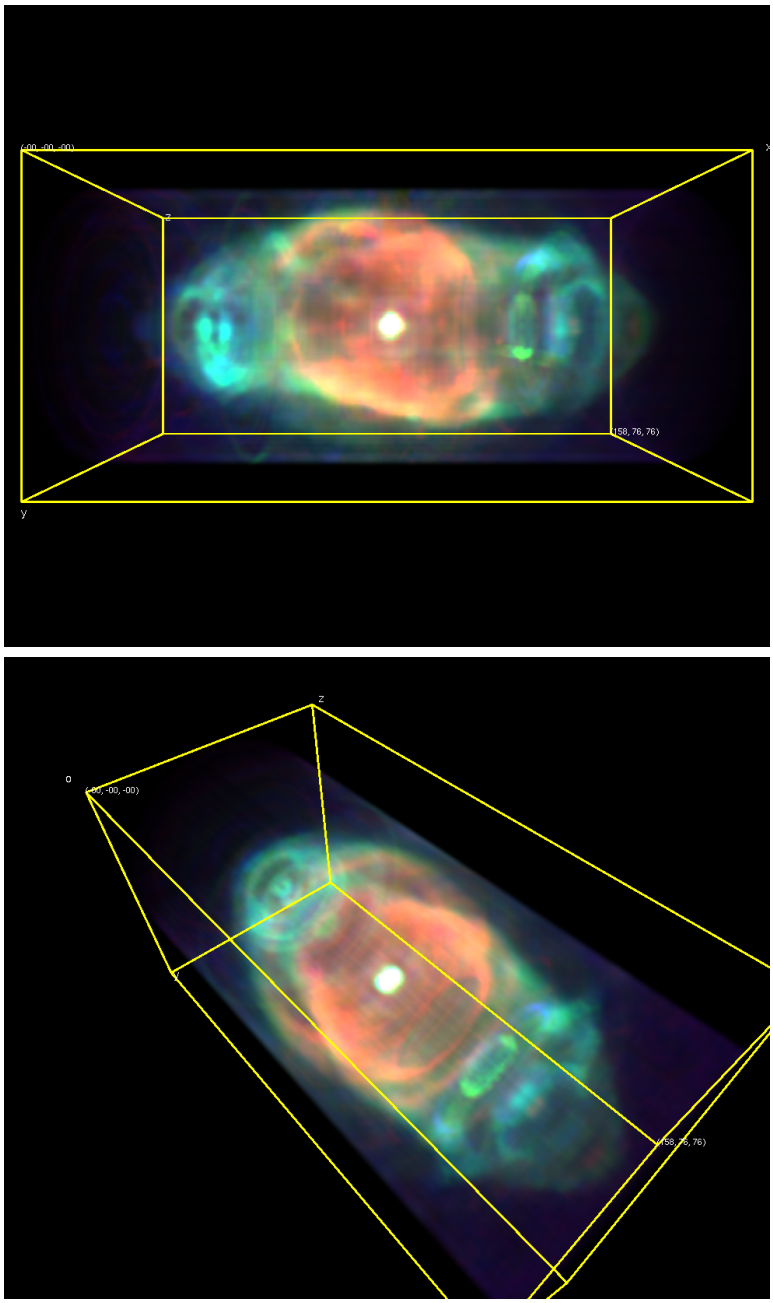

Figure 5.9: Asymmetry correction, using the proportional density rule, of the Saturn Nebula: view from earth and from outer space. The red, green and blue color channels are assigned to 658nm, 555nm and 502nm, respectively. Note how the asymmetry correction restores the S-shape of the original image.

Chapter 6

Discussion and Conclusion

6.1 Discussion

Our reconstructed results closely resemble the actual images. There are, however, some limitations in our approach. First, the inclination angle cannot be reliably determined by our algorithm. Different inclinations may yield equally credible results. Then, the asymmetry correction is not based on any physical measurement and can only heuristically and ambigously determined, as long as no additional data is provided. A similar limitation applies for the axisymmetric shape of the object that cannot be verified by the program but has to be assumed. The reconstruction is also only possible for nebulae whose axis of symmetry is not too inclined with respect to the image plane because no reliable 3D information can be derived if the axis is close to parallel to the viewing direction. Objects without symmetry cannot be reconstructed at all, for the same reason.

However, for axisymmetric objects the results can be guaranteed to resemble the original. The CGLS algorithm is guaranteed to converge, and small asymmetric features can be guaranteed to break the symmetry in such a way that the original image is exactly reconstructed. For objects with spherical symmetry, the algorithm is also applicable, even though it could be further optimized to benefit from the stricter constraints.

In order to verify the accurateness of our reconstruction, we reconstruct an artificial model with perfect axisymmetry. For this, a radial map of intensities is drawn and projected into the image plane at different inclination angles. The resulting images are then reconstructed and the reconstructed intensity maps are compared to the original. We can show that for inclinations at least up to 45 degrees, the reconstructed radial map very closely resembles the original (see fig. 6.1).

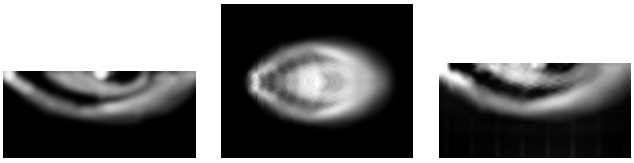

Figure 6.1: An artificial "nebula" used for testing the reconstruction: radial map, rendered view with 35 degrees inclination, reconstructed radial map (reconstructed with somewhat lower resolution and scaled to the size of the original radial map)

6.2 Conclusion

We have presented an algebraic reconstruction approach to derive 3D information from single images of axisymmetric purely emissive objects like planetary nebulae. The algorithm is able to generate either purely axisymmetric models or models with correction for small deviations from axial symmetry. The calculations are carried out efficiently by making use of the GPU's parallel computing power.

The resulting three-dimensional models can be rendered from arbitrary perspective. This allows for a wide field of applications, in scientific as well as artistic contexts. They may help understanding the complex structure of planetary nebulae and the physical mechanisms that underlie their formation.

6.3 Outlook

A possible extension of the program would be further interpolation of the created 3D models, possibly in real time, using heuristic rules. While the results would surely deviate from physical reality, artistic usage of such models would be imaginable.

Another enhancement would be the inclusion of absorption in the model so it would be able to accurately describe dusty environments as well. Since in this case the reconstruction becomes ambiguous, some other data would have to be included to guarantee physical realism.

Furthermore, simulating the ionization process inside the planetary nebula would allow to check the physical plausibility of the reconstructed model (at least for matter-bounded nebulae) by comparing the computed ionization front to the reconstructed one. The atom densities that are needed as input for the simulation could be derived from the reconstructed ion densities.

Chapter 7

Acknowledgments

This project is partially funded by the German science foundation DFG under grant 444 MEX-113/25/0-1 and the Mexican science foundation CONACyT under a bi-lateral cooperation grant.

Bibliography

[bla] Fortran BLAS library. `http://www.netlib.org/blas/`.

[cud] nvidia CUDA. `http://www.nvidia.com/object/cuda_develop.html`.

[FTI88] A. Fujimoto, Takayuki Tanaka, and K. Iwata. Arts: accelerated ray-tracing system. pages 148–159, 1988.

[geo] Geometric tools library. `http://www.geometrictools.com`. Especially `Wm4IntrLine3Cylinder3.cpp` from `/LibFoundation/Intersection/`.

[han] `netlib.org` CGLS algorithm. file `REGU/cgls.m` from `http://netlib.org/numeralgo/na4-matlab7.tgz`.

[Han96] Per Christian Hansen. *Rank-Deficient and Discrete Ill-Posed Problems*. Polyteknisk Forlag, 1996. Especially chapter 6: "Iterative methods".

[IM04] Ivo Ihrke and Marcus Magnor. Image-based tomographic reconstruction of flames. In *ACM Siggraph / Eurographics Symposium Proceedings, Symposium on Computer Animation*, pages 367–375, June 2004.

[KS88] A. C. Kak and Malcolm Slaney. *Principles of Computerized Tomographic Imaging*. IEEE Press, 1988. Especially chapter 7: "Algebraic Reconstruction Algorithms".

[MKHD04] M. Magnor, G. Kindlmann, C. Hansen, and N. Duric. Constrained inverse volume rendering for planetary nebulae. In *Proc. IEEE Visualization (Vis'04), Austin, USA*, pages 83–90, October 2004.

[MKHD05] M. Magnor, G. Kindlmann, C. Hansen, and N. Duric. Reconstruction and visualization of planetary nebulae. In *IEEE Trans. on Visualization and Computer Graphics*, volume 11/5, pages 485–496, September 2005.

[OF06] Donald E. Osterbrock and Gary J. Ferland. *Astrophysics of Gaseous Nebulae and Active Galactic Nuclei*. University Science Books, 2006.

[SCB+00] F. Sabbadin, E. Cappellaro, S. Benetti, M. Turatto, and C. Zanin. Tomography of the low excitation planetary nebula NGC 40. *Astronomy & Astrophysics*, 355, 2000.

[ubl] boost library UBLAS. `http://www.boost.org/doc/libs/1_35_0`.